МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Івано-Франківський національний технічний університет нафти і газу

Кафедра комп’ютеризованого машинобудівного виробництва

В. Б. Копей

ЯДРО ГЕОМЕТРИЧНОГО МОДЕЛЮВАННЯ OPEN CASCADE TECHNOLOGY ДЛЯ PYTHON-ПРОГРАМІСТІВ

МЕТОДИЧНІ ВКАЗІВКИ ДЛЯ САМОСТІЙНОЇ РОБОТИ

Івано-Франківськ
2017

УДК 004.925.8
К 65

Рецензент:

Панчук В. Г., доктор технічних наук, професор, завідувач кафедри комп’ютеризованого машинобудівного виробництва Івано-Франківського національного технічного університету нафти і газу

Рекомендовано методичною радою університету (протокол № 22 від 25.05.2017 р.)

К65 **Копей В. Б.** Ядро геометричного моделювання Open CASCADE Technology для Python-програмістів: Методичні вказівки для самостійної роботи / В. Б. Копей - Івано-Франківськ: ІФНТУНГ, 2017. - 47 с.

МВ 02070855-10997 - 2017

Методичні вказівки призначені для самостійного вивчення ядра геометричного моделювання Open CASCADE Technology на прикладах програм мовою Python. Наведені приклади створення та візуалізації геометричних моделей за допомогою PythonOCC та FreeCAD. Розроблено відповідно до робочої програми дисципліни "Розмірне моделювання та аналіз технологічних процесів" для підготовки магістрів за спеціальністю 131 - Прикладна механіка.

УДК 004.925.8

МВ 02070855-10997 - 2017

ЗМІСТ

ВСТУП

Ядро геометричного моделювання - це набір бібліотек з програмним інтерфейсом (API) для розробки програмних продуктів, що працюють з геометричними моделями [1-9]. В таблиці 1 перелічені поширені ядра геометричного моделювання та програмні продукти класу CAD/CAM/CAE, в яких вони використовуються.

Таблиця 1 - Поширені ядра геометричного моделювання

Назва ядра	**Розробник**	**В яких програмних продуктах використовується**
Parasolid	Siemens PLM Software	NX, Solid Edge, Femap, SolidWorks, PowerSHAPE, Abaqus, ANSYS
ACIS	Dassault Systemes	AutoCAD, SpaceClaim, MicroStation
CGM	Dassault Systemes	CATIA
Granite	Parametric Technology Corporation	PTC Creo (Pro/Engineer)
OCCT*	OPEN CASCADE S.A.S.	SALOME, FreeCAD

* вільно розповсюджується

Open CASCADE Technology 6.8.0 (OCCT) - платформа розробки програмного забезпечення, яка забезпечує сервіси для 3D поверхневого і твердотільного моделювання, обміну даними з CAD та візуалізації [10]. Більша частина функціональності OCCT доступна з C++ бібліотек. OCCT може бути застосована під час розробки програм, які працюють з 3D моделюванням (CAD), виробництвом / вимірюванням (CAM), або моделюванням чисельними методами (CAE).

Авторські права на OCCT належать французькій компанії OPEN CASCADE S.A.S. [10]. OCCT є вільним програмним забезпеченням і ліцензується за GNU LGPL 2.1 (Загальна

громадська ліцензія обмеженого використання GNU) з додатковим винятком.

Відома відкрита твердотільна CAD FreeCad [11] та відкритий пре- и постпроцесор для моделювання чисельними методами SALOME [12] розроблені на основі OCCT.

Форком (відгалуженням) OCCT 6.8.0 є C++ бібліотека Open CASCADE Community Edition 0.17 (OCE). Цей проект має мету збирати різноманітні патчі, зміни і удосконалення.

Для програмістів мовою Python [13, 14] існує вільна бібліотека pythonOCC 0.16, побудована на основі OCE 0.17 за допомогою SWIG - інструмента для пов'язування коду C++ та Python. Розробником pythonOCC є французький програміст Томас Павіот [15].

Ці методичні вказівки призначені для самостійного вивчення ядра геометричного моделювання Open CASCADE Technology на прикладах програм мовою Python. Методичні вказівки можуть бути використані студентами спеціальності "Прикладна механіка" під час вивчення дисциплін "Розмірне моделювання та аналіз технологічних процесів", "Моделювання та аналіз конструкцій машин", "Методи моделювання і симуляції кінематики і динаміки машин", під час виконання дипломних робіт, а також усіма, хто цікавиться програмуванням геометричних моделей.

ОГЛЯД OPEN CASCADE TECHNOLOGY

Класи OCCT групуються в пакети. Щоб запобігти конфлікту імен назви класів починаються з назви пакету. Пакети поділяються на бібліотеки (toolkits), які діляться модулі. Документація по класам в OCCT розділена на наступні частини:

- Foundation Classes (базові класи) [16];
- Modeling Data (дані для моделювання) [17];
- Modeling Algorithms (алгоритми для моделювання) [18];
- Mesh (сітка);
- Visualization (візуалізація);
- Data Exchange (обмін даними);
- Application Framework (каркас прикладної програми);

- Open CASCADE Test Harness (інструмент для тестування).

Базові класи (Foundation Classes) забезпечують різноманітні сервіси загального призначення. Наприклад:

- базові типи, рядки і різноманітні типи величин;
- автоматизоване управління динамічною пам'яттю;
- обробка помилок;
- класи для маніпуляцій колекціями даних;
- математичні інструменти (вектори, матриці і типи геометричних примітивів);
- базові сервіси для збереження даних у ASCII файли.

Ці класи організовані в наступні бібліотеки:

- Kernel Classes (класи ядра);
- Math Utilities (математичні утиліти);
- Basic Data Storage (базове збереження даних).

Дані для моделювання (Modeling Data) являють собою структури даних для подання 2D і 3D геометричних моделей. Ці сервіси організовані в наступні бібліотеки:

- 2D Geometry Types (2D геометричні типи);
- 3D Geometry Types (3D геометричні типи);
- Geometry Utilities (геометричні утиліти);
- Topology (топологія).

Алгоритми для моделювання (Modeling Algorithms) включають широкий діапазон топологічних алгоритмів, які використовуються в моделюванні, та геометричних алгоритмів, які викликаються ними. Ці сервіси організовані в наступні бібліотеки:

- Geometric Tools (геометричні інструменти);
- Topological Tools (топологічні інструменти);
- Construction of Primitives (створення примітивів);
- Boolean Operations (булеві операції);
- Fillets and Chamfers (заокруглення і фаски);
- Offsets and Drafts (зміщення і витягування);
- Features (елементи);
- Hidden Line Removal (видалення невидимих ліній);
- Sewing (зшивання);
- Shape Healing (виліковування форм).

Сітка (Mesh) - засоби представлення 3D об'єктів у вигляді полігональних сіток. В них входять:

- структури даних для збереження даних поверхневих сіток, які асоційовані з формами, і деякі базові алгоритми для обробки цих даних;

- структури даних і алгоритми для побудови поверхневих трикутних сіток з BRep об'єктів (форм);

- інструменти для розширення можливостей 3D візуалізації з відображенням сіток разом з асоційованими даними пре- і пост-процесора.

Додатково існують інструменти для експорту моделей в сіткові формати VRML та STL.

Візуалізація (Visualization) в Open CASCADE Technology основана на розділенні даних моделей, які ви хочете показати і вибрати, і на графічному представленні їх структури.

Для візуалізації структур даних OCCT має готові алгоритми, які створюють графічне представлення з геометричних моделей. Ці структури даних можуть бути використані з переглядачами, що постачаються, або налаштовані з врахуванням специфіки ваших програм.

Відображення управляється через презентаційні сервіси, а вибір управляється через сервіси вибору. З цими сервісами представлені структури даних і алгоритми для відображення об'єктів програми і підтримки графічного вибору цих об'єктів.

Інтерактивні сервіси застосування (Application Interactive Services) представлені для управління відображенням, виявленням і вибором графічної презентації. Ці сервіси асоційовані з структурами даних і інтерактивними об'єктами.

Інтерфейси обміну даними (Data Exchange) в OCCT дозволяють програмі обмінюватись даними з різноманітними САПР, що забезпечує високий рівень сумісності.

Інтерфейси стандартизованого обміну даними підтримують формати STEP (AP203: проектування машин, що включає загальний 3D CAD; AP214: проектування автомобілів), IGES (до 5.3), сіткові формати VRML та STL.

Розширений обмін даними дозволяє розширити діапазон обміну шляхом передачі додаткових даних, які прикріплені до геометричних даних BREP.

Компоненти обміну даними високого рівня додаються до компонентів розширеного обміну даними та підтримують формати ACIS SAT, Parasolid, DXF.

Каркас прикладної програми (Application Framework) забезпечує рішення для обробки даних застосувань на основі парадигми застосування/документ. Він використовує асоціативний рушій для розробки CAF застосувань за рахунок наступних функцій:

- Атрибути даних забезпечують управління даними програми. Атрибути можуть бути організовані відповідно до потреб розробника;

- Сервіси збереження даних і персистентності;

- Можливість модифікації і переобчислення документів;

- Можливість управління багатьма документами;

- Готові для використання атрибути даних моделювання, які є загальними для CAD/CAM програм;

- Готові для використання функції відмінити-повернути і скопіювати-вставити.

Інструмент для тестування Open CASCADE Test Harness, який також відомий як DRAW (DRAW.exe) - це легкий у використанні командний інтерпретатор на основі мови Tcl і графічної системи, який призначений для тестування і демонстрації бібліотек OCCT. DRAW може бути використаний інтерактивно для створення, відображення та модифікації таких об'єктів як криві, поверхні та топологічні форми.

РЕКОМЕНДАЦІЇ ДЛЯ ВИВЧЕННЯ OCCT ТА PYTHONOCC

1 Скачайте і установіть pythonOCC-0.16.2-win32-py27.exe [15].
2 Скачайте вихідні коди з прикладами: https://github.com/tpaviot /pythonocc-core/archive/master.zip. Ознайомтесь з прикладами мовою Python.
3 Виконайте базовий урок (occt_tutorial.pdf).

4 Ознайомтесь з базовою документацією [16, 17, 18].
5 Після установки OCCT [10] використовуйте документацію: opencascade-6.8.0\doc\refman\index.html та opencascade-6.8.0\doc \overview\index.html.
6 Для довідки по класам переглядайте вміст файлів cdl з каталогу opencascade-6.8.0\src.
7 Приклади шукайте в каталозі opencascade-6.8.0\samples.
8 Експериментуйте з DRAW Test Harness (occt_test_harness.pdf). Приклади мовою Tcl знаходяться в каталозі opencascade-6.8.0\tests, а приклади мовою C++ шукайте в його вихідних кодах (opencascade-6.8.0\src). Це каталоги TestTopOpeDraw, TestTopOpeTools, TestTopOpe, BRepTest, GeometryTest, HLRTest, MeshTest, GeomliteTest, DrawFairCurve, BOPTest, SWDRAW, ViewerTest.
9 Переглядайте вихідні коди інших програм, розроблених за допомогою pythonOCC, наприклад, https://github.com/charles-sharman/ccad, https://code.google.com/p/caddd.
10 Використовуйте документацію pythonOCC в атрибутах __doc__, підказку коду та автодоповнення коду Python. Наприклад інтерактивна підказка коду є у Eclipse PyDev. Для автодоповнення коду в PyDev використовуйте Ctrl+Space.

ПРИКЛАД 1. НАЙПРОСТІША ПРОГРАМА

У цьому прикладі (test_simpleBox.py) створюється форма призми (рис. 1) і показується на екрані за допомогою засобів створення простого графічного інтерфейсу користувача (GUI). Надалі у прикладах символами ···· позначається Python-відступ, під яким потрібно розуміти чотири пробіли.

```
# encoding: utf-8
# Імпортувати функцію для створення простого GUI
from OCC.Display.SimpleGui import init_display
display, start_display, add_menu,
add_function_to_menu = init_display()
```



```
# Імпортувати функцію для створення призми
from OCC.BRepPrimAPI import BRepPrimAPI_MakeBox
my_box = BRepPrimAPI_MakeBox(10., 20.,
30.).Shape() # створити призму
display.DisplayShape(my_box,update=True) #
показати призму
start_display() # цикл обробки повідомлень. Цей
виклик повинен бути останнім.
```

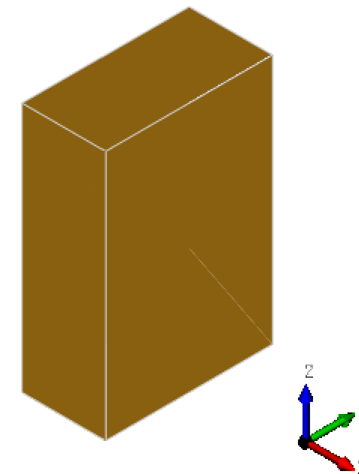


Рисунок 1 - Приклад 1

ПРИКЛАД 2. ІМПОРТ НАЙБІЛЬШ УЖИВАНИХ МОДУЛІВ PYTHONOCC

У прикладі розроблено модуль myBaseGeom.py, який імпортує модулі pythonOCC та створює об'єкти, що будуть використовуватись в усіх наступних прикладах. Щоб не повторювати цей код, наступні приклади виконують імпорт цього модуля командою `import myBaseGeom`.

```
# encoding: utf-8
from OCC import VERSION # версія PythonOCC
```

```
print VERSION

# Засоби для створення простого GUI
from OCC.Display.SimpleGui import init_display
display, start_display, add_menu,
add_function_to_menu = init_display()

# Геометричний процесор gp - незбережувані
базові геометричні об'єкти
# З цими об'єктами працюють за значенням
from OCC.gp import *

# Збережувані базові 3D геометричні об'єкти
# Ці об'єкти STEP-оброблювані і з ними працюють
за посиланням
from OCC.Geom import *
# Збережувані базові 2D геометричні об'єкти
# Ці об'єкти STEP-оброблювані і з ними працюють
за посиланням
from OCC.Geom2d import *

# Алгоритми для побудови елементарних
геометричних об'єктів OCC.Geom
from OCC.GC import *
# Алгоритми для побудови елементарних
геометричних об'єктів OCC.Geom2d
from OCC.GCE2d import *
```

ПРИКЛАД 3. ЕЛЕМЕНТАРНІ ГЕОМЕТРИЧНІ ОБ’ЄКТИ

У прикладі test_gp.py демонструється геометричний процесор gp - незбережувані базові геометричні об'єкти. З цими об'єктами працюють за значенням. Демонструються також алгоритми для побудови елементарних геометричних об'єктів gce.

```
# encoding: utf-8
```



```
from myBaseGeom import *

p1=gp_Pnt(0,0,0) # 3D точка в декартовій системі
координат
print p1.X() # координата X
print p1.Distance(gp_Pnt(1,1,1)) # відстань між
точками
p2=p1.Translated(gp_Pnt(0,0,0),gp_Pnt(1,0,0)) #
нова точка шляхом переміщення

vec1=gp_Vec(gp_Pnt(0,0,0), gp_Pnt(1, 1, 1)) # 3D
вектор
print vec1.Magnitude() # довжина вектора
vec2=gp_Vec(gp_Pnt(0,0,0), gp_Pnt(1, 0, 0)) # 3D
вектор
print vec1.Angle(vec2) # кут між векторами

dir1=gp_Dir(vec1) # одиничний вектор в 3D
просторі (напрямок)
dir2=gp_Dir() # напрямок, який відповідає осі X
dir2=gp_Dir(1,0,0) # або так
dir2=gp_DX() # або так

ax1=gp_Ax1(p1, dir1) # вісь в 3D просторі
ax2=gp_Ax1() # вісь Z системи координат

ax3=gp_Ax2(p1,dir2) # правостороння система
координат в 3D просторі (dir2 задає головний
напрямок)
ax4=gp_Ax2() # вихідна система координат (OXYZ)
ax5=gp_Ax3(ax3) # система координат в 3D
просторі, яка може бути правосторонньою або
лівосторонньою

mat1=gp_Mat() # нульова матриця для векторних і
матричних обчислень
```

```
ln1=gp_Lin(p1, dir1) # лінія в 3D просторі
print ln1.Location().X() # X координата вихідної точки лінії
print ln1.Distance(p2) # відстань до точки
ln2=ln1.Normal(p2) # лінія, нормальна до заданої

cir1=gp_Circ(ax4,5) # коло в 3D просторі радіусом 5

pln1=gp_Pln(p1, dir1) # площина (dir1 нормальний до площини)
pln2=gp_Pln() # площина OXY
print pln2.XAxis() # вісь X площини

sph1=gp_Sphere(ax5,5) # сфера з локальною системою координат ax5 і радіусом 5

cyl1=gp_Cylinder(ax5,5) # циліндр з локальною системою координат ax5 і радіусом 5

trsf1=gp_Trsf() # визначає незбережувану трансформацію в 3D просторі
trsf1.SetTranslation(vec1) # змінює трансформацію на переміщення
# див. також BRepBuilderAPI_Transform

###############################################
# Для 2D існують аналогічні класи, але містять вкінці "2d", наприклад:
# gp_Pnt2d, gp_Vec2d, gp_Dir2d ,gp_Ax22d

###############################################
# Алгоритми для побудови елементарних геометричних об'єктів OCC.gp
from OCC.gce import *
```



```
ln3=gce_MakeLin(p1,p2) # алгоритм побудови лінії
if ln3.IsDone(): # якщо побудова успішна
····ln3=ln3.Value() # лінія класу OCC.gp.gp_Lin
else:
····ln3.Status() # статус помилки
trsf2=gce_MakeTranslation(vec1) # алгоритм трансформації-переміщення
trsf2=trsf2.Value() # трансформація-переміщення класу gp_Trsf
display.DisplayShape(p1)
start_display()
```

ПРИКЛАД 4. ЗБЕРЕЖУВАНІ БАЗОВІ 3D ГЕОМЕТРИЧНІ ОБ'ЄКТИ

У прикладі test_Geom.py розглядається модуль Geom, який містить засоби створення збережуваних базових 3D геометричних об'єктів (рис. 2). Ці об'єкти STEP-оброблювані і з ними працюють за посиланням.

```
# encoding: utf-8
from myBaseGeom import *
p1=gp_Pnt(0,0,0) # 3D точка в декартовій системі координат
p2=gp_Pnt(1,1,0)
p3=gp_Pnt(1,0,0)
ln1=GC_MakeLine(p1,p2).Value() # лінія (OCC.Geom.Handle_Geom_Line)
ln2=GC_MakeSegment(p1,p2).Value() # відрізок (OCC.Geom.Handle_Geom_TrimmedCurve)
ln3=GC_MakeSegment(gp_Lin(),gp_Pnt(),gp_Pnt(0,0,1)).Value() # відрізок за точками на лінії
ln3=GC_MakeSegment(gp_Lin(),0,1).Value() # відрізок за двома значеннями параметра лінії
cir1=GC_MakeCircle(p1,p2,p3).Value() # коло (OCC.Geom.Handle_Geom_Circle)
```

```
cir2=Geom_Circle(gp_XOY(),1).GetHandle() # коло
(OCC.Geom.Handle_Geom_Circle)
arc1=GC_MakeArcOfCircle(p1,p2,p3).Value() # дуга
(OCC.Geom.Handle_Geom_TrimmedCurve)
tra1=GC_MakeTranslation(p1,p2).Value() #
переміщення
(OCC.Geom.Handle_Geom_Transformation)
display.View_Top() # вид зверху
# Нарисувати фігури
for s in [p1,p2,p3,ln2,cir2,arc1]:
····display.DisplayShape(s,update=True,color="bl
ack")
# перевірка успішності побудови
cir1=GC_MakeCircle(p1,p2,p3)
if cir1.IsDone(): # якщо побудова успішна
····cir1=cir1.Value() # коло
else:
····print cir1.Status() # статус помилки
start_display()
```

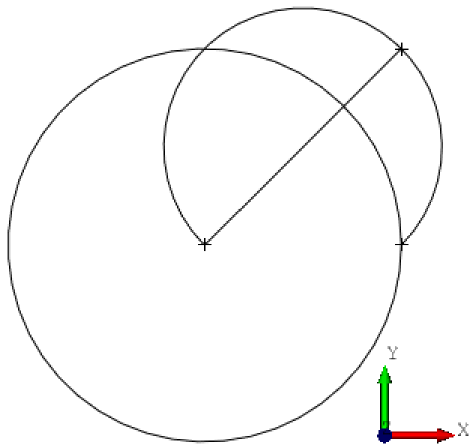


Рисунок 2 - Приклад 4

ПРИКЛАД 5. ЗБЕРЕЖУВАНІ БАЗОВІ 2D ГЕОМЕТРИЧНІ ОБ'ЄКТИ

У прикладі test_Geom2d.py розглядається модуль Geom2D, який містить засоби створення збережуваних базових 2D геометричних об'єктів (рис. 3). Ці об'єкти STEP-оброблювані і з ними працюють за посиланням.

```
# encoding: utf-8
from myBaseGeom import *

p1=gp_Pnt2d(0,0)
p2=gp_Pnt2d(1,0)
p3=gp_Pnt2d(1,-1)
vec1=gp_Vec2d(p3,p1)
ln1=GCE2d_MakeSegment(p3,p1).Value() # відрізок
(OCC.Geom2d.Handle_Geom2d_TrimmedCurve)
# перетворення об'єктів gp в об'єкти Geom2d:
ln2=Geom2d_Line(gp_Lin2d()) # лінія
(OCC.Geom2d.Geom2d_Line)
arc2=GCE2d_MakeArcOfCircle(p1,vec1,p2).Value() #
дуга через дві точки та дотична до vec1 в точці
p1 (OCC.Geom2d.Handle_Geom2d_TrimmedCurve)
offs1 = Geom2d_OffsetCurve(arc2, -0.1) # крива
отримана шляхом зміщення
OCC.Geom2d.Geom2d_OffsetCurve

display.View_Top() # вид зверху
# Нарисувати фігури
for s in [p1,p2,p3,ln1,ln2,arc2,offs1]:
····display.DisplayShape(s,color="black")

from OCC.GeomAPI import geomapi
arc3=geomapi.To3d(arc2 ,gp_Pln()) # перетворити
2D в 3D (OCC.Geom.Handle_Geom_Curve)
```

```
arc4=geomapi.To2d(arc3 ,gp_Pln()) # перетворити 3D в 2D (OCC.Geom2d.Handle_Geom2d_Curve)

display.FitAll()
start_display()
```

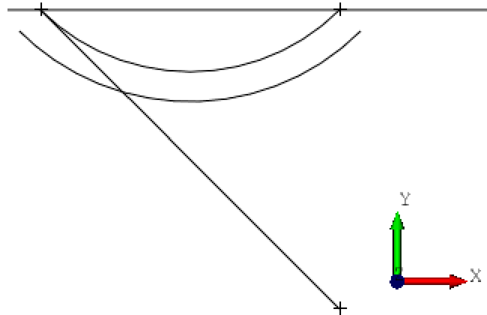


Рисунок 3 - Приклад 5

ПРИКЛАД 6. ГЕОМЕТРИЧНІ ОБМЕЖЕННЯ ОБ'ЄКТІВ GP

У прикладі test_GccAna.py демонструються аналітичні алгоритми для створення об'єктів з геометричними обмеженнями для об'єктів gp (рис. 4).

```
# encoding: utf-8
from myBaseGeom import *
from OCC.GccEnt import gccent # об'єкти з кваліфікаторами для створення об'єктів з геометричними обмеженнями
from OCC.GccAna import * # аналітичні алгоритми для створення об'єктів з геометричними обмеженнями для об'єктів gp

ax=gp_Ax22d() # система координат OXY
```



```
cir=gp_Circ2d(ax,5) # коло радіусом 5 (OCC.gp.gp_Circ2d)
ptn=gp_Pnt2d(7,7) # точка

cire=gccent.Outside(cir) # коло з кваліфікатором (GccEnt_QualifiedCirc)

# Спробуйте різні кваліфікатори, щоб побачити різні результати:
# Enclosing - розв'язок (лінія) повинен обводити аргумент (коло)
# Enclosed - розв'язок повинен бути обведений аргументом
# Outside - розв'язок і аргумент повинні бути назовні один до одного
# Unqualified - позиція невизначена
# Примітка: в даному випадку кваліфікатор Enclosed не допустимий
# а кваліфікатор Unqualified дає два розв'язки

sol=GccAna_Lin2d2Tan(cire, ptn, 1e-6) # алгоритм побудови лінії через точку і яка дотична до кола
print sol.NbSolutions() # кількість розв'язків
ln=sol.ThisSolution(1) # перший розв'язок - лінія (OCC.gp.gp_Lin2d)

display.View_Top() # вид зверху
# Нарисувати фігури
for s in [Geom2d_Line(ln),ptn,Geom2d_Circle(cir)]:
    display.DisplayShape(s,update=True,color="black")

start_display()
```

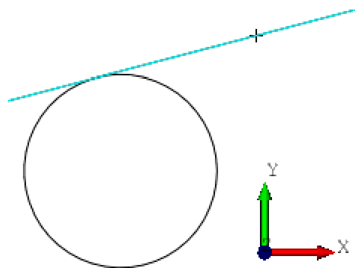


Рисунок 4 - Приклад 6

ПРИКЛАД 7. ГЕОМЕТРИЧНІ ОБМЕЖЕННЯ ОБ'ЄКТІВ GEOM2D

У прикладі test_Geom2dGcc.py демонструється модуль Geom2dGcc, який містить алгоритми для створення об'єктів з геометричними обмеженнями для об'єктів Geom2d (рис. 5).

```
# encoding: utf-8
from myBaseGeom import *
from OCC.GccEnt import * # об'єкти з кваліфікаторами для створення об'єктів з геометричними обмеженнями
from OCC.Geom2dGcc import * # алгоритми для створення об'єктів з геометричними обмеженнями для об'єктів Geom2d
from OCC.Geom2dAdaptor import * # адаптери для об'єктів Geom2d
from OCC.Precision import precision_Angular # рекомендована точність для перевірки рівності двох кутів (1e-12)
cir=gp_Circ2d(gp_Ax22d(),5) # коло радіусом 5 (OCC.gp.gp_Circ2d)
circ=GCE2d_MakeCircle(cir).Value() # коло (OCC.Geom2d.Handle_Geom2d_Circle)
ptn=gp_Pnt2d(6,4) # точка
```



```
# адаптує криву Geom2d для її використання в алгоритмах
adap=Geom2dAdaptor_Curve(circ) # адаптер кривої (OCC.Geom2dAdaptor.Geom2dAdaptor_Curve)
# Спробуйте різні кваліфікатори, щоб побачити різні результати:
# 0 - Unqualified
# 1 - Enclosing
# 2 - Enclosed
# 3 - Outside
cire=Geom2dGcc_QualifiedCurve(adap,0) # коло з кваліфікатором 0 (OCC.Geom2dGcc.Geom2dGcc_QualifiedCurve)
sol=Geom2dGcc_Lin2d2Tan(cire,ptn,precision_Angular()) # алгоритм побудови лінії через точку і яка дотична до кола
print sol.NbSolutions() # кількість розв'язків
ln=sol.ThisSolution(1) # перший розв'язок - лінія (OCC.gp.gp_Lin2d)
display.View_Top() # вид зверху
# Нарисувати фігури
for s in [circ,ptn,Geom2d_Line(ln)]:
    display.DisplayShape(s,update=True,color="black")
start_display()
```

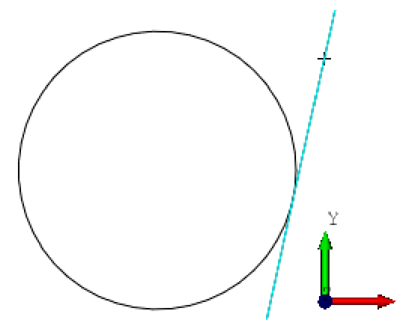


Рисунок 5 - Приклад 7

ПРИКЛАД 8. ПОБУДОВА B-СПЛАЙНА

У прикладі демонструється побудова B-сплайна (рис. 6) за допомогою Geom2dAPI_PointsToBSpline.

```
# encoding: utf-8
from myBaseGeom import *

from OCC.Geom2dAPI import
Geom2dAPI_PointsToBSpline
# можна також використати Geom2dAPI_Interpolate
from OCC.TColgp import TColgp_Array1OfPnt2d
array = TColgp_Array1OfPnt2d(1, 5) # масив точок
розміром 5

for i,p in enumerate([(0, 1),(1, 2),(2, 3),(3,
3),(4, 4)]): # для кожної точки
····array.SetValue(i+1, gp_Pnt2d(*p)) #
помістити точку в масив

bspl1 = Geom2dAPI_PointsToBSpline(array).Curve()
# сплайн

display.View_Top() # вид зверху
display.DisplayShape(bspl1,update=True,color='bl
ack') # показати сплайн

for i in range(array.Lower(), array.Upper()+1):
# для кожної точки
····display.DisplayShape(array.Value(i),color='b
lack',update=False) # показати точку

start_display()
```

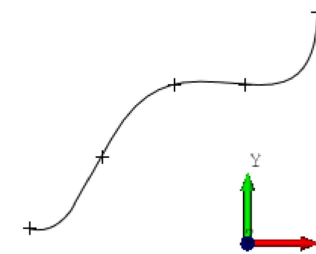



Рисунок 6 - Приклад 8

ПРИКЛАД 9. ПЕРЕТИН ДВОХ КРИВИХ

У прикладі test_InterCurveCurve.py показано побудову точок перетину двох кривих (рис. 7) за допомогою Geom2dAPI_InterCurveCurve.

```
# encoding: utf-8
from myBaseGeom import *
from OCC.Geom2dAPI import Geom2dAPI_InterCurveCurve
ln=GCE2d_MakeLine(gp_Pnt2d(0,0),gp_Pnt2d(1,1)).V
alue() # лінія
cir=GCE2d_MakeCircle(gp_Pnt2d(0,0),5).Value() # коло
inter=Geom2dAPI_InterCurveCurve(ln, cir) #
алгоритм перетину двох кривих
n=inter.NbPoints() # кількість перетинів
m=inter.NbSegments() # кількість тангенціальних
перетинів
for i in range(1,n+1): # для кожного перетину
····p=inter.Point(i) # точка перетину
····display.DisplayShape(p,color='black') #
показати точку перетину
display.View_Top() # вид зверху
display.DisplayShape(cir,color='black')
display.DisplayShape(ln,color='black')
start_display()
```

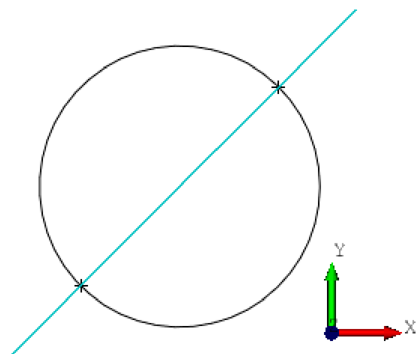


Рисунок 7 - Приклад 9

ПРИКЛАД 10. ОРТОГОНАЛЬНА ПРОЕКЦІЯ ТОЧКИ НА КРИВУ

У прикладі test_ProjectPointOnCurve.py показано побудову ортогональної проекцій точки на криву (рис. 8) за допомогою Geom2dAPI_ProjectPointOnCurve.

```
# encoding: utf-8
from myBaseGeom import *

from OCC.Geom2dAPI import
Geom2dAPI_ProjectPointOnCurve
cir=GCE2d_MakeCircle(gp_Pnt2d(0,0), 5).Value() #
коло
ptn=gp_Pnt2d(6,4) # точка
proj = Geom2dAPI_ProjectPointOnCurve(ptn,cir) #
ортогональна проекція точки на криву
n=proj.NbPoints() # кількість проекції

for i in range(1,n+1): # для кожної проекції
····p=proj.Point(i) # точка проекції
····display.DisplayShape(p,color='black') #
показати точку проекції
```



```
····dist = proj.Distance(i) # відстань від ptn
до p
····display.DisplayMessage(p,"Distance:
%f"%dist,message_color=(0,0,0))

print proj.NearestPoint() # найближча точка
проекції
print proj.LowerDistance() # найменша відстань
до проекції

display.View_Top() # вид зверху
display.DisplayShape(ptn,color='black')
display.DisplayShape(cir,color='black',update=Tr
ue)

start_display()
```

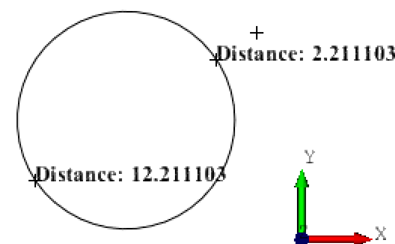


Рисунок 8 - Приклад 10

ПРИКЛАД 11. ТОПОЛОГІЧНІ ОБ'ЄКТИ

У прикладі test_TopoDS.py описані ресурси для топологічно орієнтованих застосувань, наприклад, перерахування форм, орієнтації форм та позиції точки відносно форми.

OCCT-топологія (рис. 9) дозволяє доступ і маніпуляцію об'єктами без їх 2D чи 3D представлення. В той час як OCCT-геометрія забезпечує описи об'єктів в термінах координат чи

параметричних значень, OCCT-топологія описує структури даних об'єктів в параметричному просторі. Ці описи використовують розташування і обмеження частин цього простору. Для забезпечення цих описів абстрактна топологія OCCT дає наступні сервіси:

- відслідковування розташування форм;
- іменування форм, субформ, їх орієнтацій і станів;
- маніпуляція формами і субформами;
- дослідження топологічних структур даних;
- використання списків і словників форм.

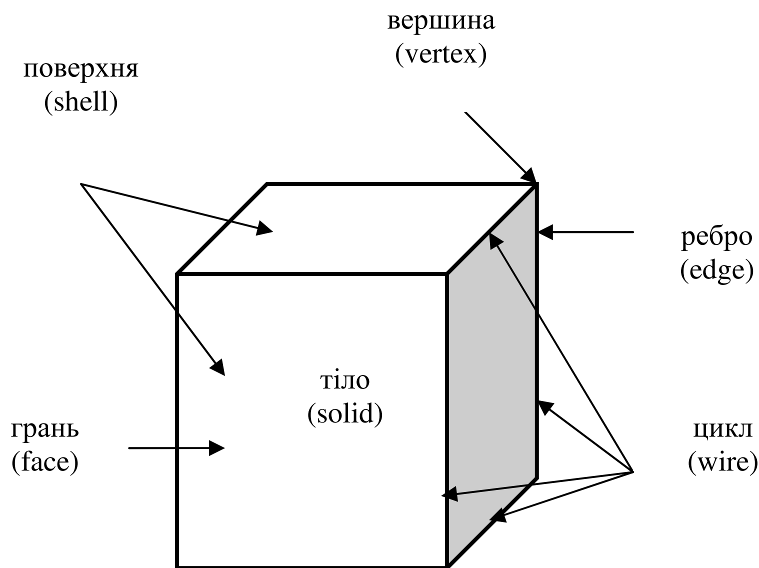


Рисунок 9 - Топологія OCCT

```
# encoding: utf-8
from myBaseGeom import *
```



```
# Ресурси для топологічно орієнтованих
застосувань,
# наприклад, перерахування форм, орієнтації форм
та позиції точки відносно форми
from OCC.TopAbs import *

# Ідентифікація різних топологічних форм (від
складного до простого):
ShapeEnum = {
····0: 'COMPOUND', # група будь-яких форм
····1: 'COMPSOLID', # група тіл з'єднаних
гранями
····2: 'SOLID', # тіло обмежене оболонкою
····3: 'SHELL', # оболонка – група граней
з'єднаних ребрами
····4: 'FACE', # грань – частина площини або
поверхня, обмежена контуром
····5: 'WIRE', # контур – послідовність ребер,
з'єднаних вершинами
····6: 'EDGE', # ребро – одновимірна форма, яка
відповідає кривій і обмежена вершинами
····7: 'VERTEX', # вершина – нуль-вимірна форма,
яка відповідає точці
····8: 'SHAPE' # форма взагалі
} # Будь-яка форма може містити простіші форми.

# Орієнтація топологічної форми – це загальне
поняття у змісті напрямку.
# Орієнтація може відображати відношення між
двома об'єктами. Використовується коли форма є
границею області і тому близьке до поняття
границі.
# Наприклад, для кривої обмеженою вершиною,
регіоном за замовчуванням є множина точок з
параметрами більшими за вершину.
```

```
# Тоді частина кривої після вершини слідує природному напрямку вздовж кривої (FORWARD).
# Для грані обмеженою ребром регіон за замовчуванням є зліва від ребра, яке слідує природному напрямку (FORWARD).
Orientation = {
····0: 'FORWARD', # матеріал є регіоном за замовчуванням (наприклад, ребро грані має орієнтацію FORWARD)
····1: 'REVERSED', # матеріал є регіоном додатковим регіону за замовчуванням (наприклад, ребро грані має орієнтацію REVERSED)
····2: 'INTERNAL', # границя лежить в матеріалі (наприклад, ребро всередині грані)
····3: 'EXTERNAL' # границя лежить поза матеріалом (наприклад, ребро поза гранню)
}

# Стан - ідентифікує позицію вершини або вершин відносно регіону форми
State = {
····0: 'IN', # точка в матеріалі
····1: 'OUT', # точка поза матеріалом
····2: 'ON', # точка на границі
····3: 'UNKNOWN' # стан точки не визначений
}

# Класи для моделювання і побудови чисто типологічних структур даних:
# TopoDS_Shape, TopoDS_Vertex, TopoDS_Edge, TopoDS_Wire, TopoDS_Face,
# TopoDS_Shell, TopoDS_Solid, TopoDS_CompSolid, TopoDS_Compound
# Забезпечує методи для перетворення об'єктів класу TopoDS_Shape в об'єкти більш спеціалізованих підкласів
```



```
from OCC.TopoDS import *

# Інструменти для топологічних структур даних, наприклад, колекції форм
from OCC.TopTools import *

from OCC.BRepBuilderAPI import *

p1=gp_Pnt(0, 0, 0) # точка
p2=gp_Pnt(0, 2, 0) # точка
medge1=BRepBuilderAPI_MakeEdge(p1,p2) # створити ребро
edge1=medge1.Shape() # ребро
vert1=medge1.Vertex1() # перша вершина
vert2=medge1.Vertex2() # друга вершина
print ShapeEnum[edge1.ShapeType()] # тип форми (TopAbs_ShapeEnum)
print Orientation[edge1.Orientation()] # орієнтація (TopAbs_Orientation)
print Orientation[vert1.Orientation()]
print Orientation[vert2.Orientation()]
print edge1.Location() # локальна система координат форми (TopLoc_Location)
print vert1.Location()
print vert2.Location()
print vert1.IsEqual(vert2) # чи ідентичні ShapeType, Orientation і Location ?

# ресурси для обробки 3D локальних систем координат
# Datum3D описує елементарну систему координат
# Location описує серію елементарних координат
from OCC.TopLoc import *
trsf=gp_Trsf() # трансформація
trsf.SetTranslation(gp_Vec(gp_Pnt(),gp_Pnt(1,0,0))) # зміщення
```

```
loc=TopLoc_Location(trsf) # створити локальну систему координат
edge1.Location(loc) # установити нову локальну систему координат форми
# або можна застосувати BRepBuilderAPI_Transform

display.DisplayShape(edge1)
display.DisplayShape(vert1)
display.DisplayShape(vert2)

display.FitAll()
start_display()
```

ПРИКЛАД 12. ТОПОЛОГІЧНИЙ API

На прикладі test_BRepAPI.py продемонстрований топологічний API - створення стандартних топологічних об'єктів, примітивів, булеві операції, заокруглення (рис. 10).

```
# encoding: utf-8
import math
from myBaseGeom import *
# Класи для моделювання і побудови чисто типологічних структур даних
from OCC.TopoDS import *
# Пакет BRep описує топологічні структури даних BRep
# (Boundary Representation Data Structure) успадкованих від абстрактної
# топології, яка визначена в пакеті TopoDS
# Ці більш складні структури об'єднують топологічні описи з додатковою геометричною інформацією
```



```
# і включають правила оцінки еквівалентності різних можливих образів одних і тих самих об'єктів, наприклад, точок.
# from OCC.BRep import *

# Забезпечує API до топологічних структур даних BRep.
# Високорівневі і прості виклики для найбільш типових операцій – створення
# вершин, ребер, граней, тіл; операції витягування, булеві операції, розрахунок глобальних властивостей
from OCC.BRepBuilderAPI import *

p1=gp_Pnt(1, 0, 0) # точка
p2=gp_Pnt(1, 2, 0) # точка
p3=gp_Pnt(2, 1, 0) # точка
mvert1=BRepBuilderAPI_MakeVertex(p1) # створює вершину (BRepBuilderAPI_MakeVertex)
vert1=mvert1.Vertex() # вершина (TopoDS_Vertex)
# але mvert1.Shape() # форма взагалі (TopoDS_Shape)
vert2=BRepBuilderAPI_MakeVertex(p2).Vertex() # вершина (TopoDS_Vertex)
edge1=BRepBuilderAPI_MakeEdge(vert1,vert2).Edge() # ребро (TopoDS_Edge)
# BRepBuilderAPI_MakeEdge(p1,p2).Edge() # або так
arc=GC_MakeArcOfCircle(p1,p3,p2).Value() # дуга
medge2=BRepBuilderAPI_MakeEdge(arc) # створює ребро з дуги
# Більшість алгоритмів підтримують обробку помилок. Наприклад:
if medge2.IsDone(): # якщо побудова успішна
    edge2=medge2.Edge() # ребро
else: # якщо помилка
```

```
····print medge2.Error() # вивести номер помилки
····# наприклад, якщо параметри функції
BRepBuilderAPI_MakeEdge будуть такі:
····# gp_Lin(),gp_Pnt(),gp_Pnt()
····# то буде істинним вираз:
····#
medge2.Error()==BRepBuilderAPI_LineThroughIdenti
cPoints

# для розрахунку екстремумів (відстаней від
форми до форми)
from OCC.BRepExtrema import BRepExtrema_ExtPC
vx=BRepBuilderAPI_MakeVertex(gp_Pnt(-2.5, 1,
0)).Vertex() # вершина
extr=BRepExtrema_ExtPC(vx,edge1) # знаходить
відстані від вершини до ребра

if extr.IsDone(): # якщо екстремуми знайдені
····print "NbExt=",extr.NbExt() # кількість
екстремумів
····print extr.SquareDistance(1) # квадрат
відстані до першого екстремуму
····print extr.IsMin(1) # чи це мінімальна
відстань?
····print extr.Parameter(1) # параметри кривої
····print extr.Point(1) # точка на кривій

mw=BRepBuilderAPI_MakeWire() # створює контур
(BRepBuilderAPI_MakeWire)
mw.Add(edge1) # додати ребро
mw.Add(edge2) # додати ребро
wire=mw.Wire() # контур (TopoDS_Wire)

# Інструменти для структур даних BRep
from OCC.BRepTools import *
```



```
wexp=BRepTools_WireExplorer(wire) # переглядач
структур TopoDS_Wire
while wexp.More(): # поки є елементи
····print wexp.Current() # поточне ребро
(TopoDS_Edge)
····print wexp.CurrentVertex() # поточна вершина
(TopoDS_Vertex)
····wexp.Next() # наступний елемент

# забезпечує API для створення примітивів
(призм, тіл обертання, витягувань, сфер,
циліндрів ...)
from OCC.BRepPrimAPI import *
face=BRepBuilderAPI_MakeFace(wire).Face() #
грань (TopoDS_Face)

# дозволяє визначити положення точки відносно
форми
# в даному випадку точка може бути "на границі"
(в грані) і "за формою"
from OCC.BRepClass3d import
BRepClass3d_SolidClassifier
sc=BRepClass3d_SolidClassifier(face,gp_Pnt(1, 1,
1),1e-6)
print "State=", sc.State() # див. OCC.TopAbs

vector=gp_Vec(p1, gp_Pnt(1, 0, 1)) # вектор
solid1 = BRepPrimAPI_MakePrism(face,
vector).Shape() # призма (TopoDS_Shape)

# Ідентифікувати позицію точки відносно грані
(на грані, поза гранню, на границі)
from OCC.BRepTopAdaptor import
BRepTopAdaptor_FClass2d
```

```
print BRepTopAdaptor_FClass2d(face,1e-6).Perform(gp_Pnt2d(1,1)) # позиція точки (TopAbs_State)
axis=gp_Ax1(gp_Pnt(),gp_Dir(0,1,0)) # вісь Y
solid2 = BRepPrimAPI_MakeRevol(face, axis, math.pi).Shape() # тіло обертання (TopoDS_Shape)

solid3 = BRepPrimAPI_MakeBox(1, 1, 1).Shape() # призма (TopoDS_Shape)
# Див. також:
# BRepPrimAPI_MakeCone # конус
# BRepPrimAPI_MakeCylinder # циліндр
# BRepPrimAPI_MakeSphere # сфера
# BRepPrimAPI_MakeTorus # тор
# BRepPrimAPI_MakeWedge # клиноподібне тіло

solid4=BRepBuilderAPI_Copy(solid3).Shape() # копія тіла (TopoDS_Shape)

trsf=gp_Trsf() # трансформація
trsf.SetTranslation(gp_Pnt(),gp_Pnt(0.5,0.5,0))
# переміщення
# Див. також:
# trsf.SetMirror() # дзеркальна трансформація
# trsf.SetScale() # масштабування
# trsf.SetRotation() # поворот
solid5=BRepBuilderAPI_Transform(solid4, trsf).Shape() # переміщена призма (TopoDS_Shape)

# забезпечує новий API для булевих операцій з формами (об'єднань, вирізів, перетинів)
from OCC.BRepAlgoAPI import *

solid6=BRepAlgoAPI_Fuse(solid1, solid2).Shape()
# тіло після об'єднання (TopoDS_Shape)
# Див. також:
```



```
# BRepAlgoAPI_Common # спільне тіло
# BRepAlgoAPI_Cut # виріз тілом
# BRepAlgoAPI_Section # перетин

# Базові інструменти для дослідження топологічних структур даних. Наприклад, дозволяє знайти усі грані тіла
from OCC.TopExp import *
# Ресурси для топологічно орієнтованих застосувань
from OCC.TopAbs import *
# API для створення заокруглень і фасок
from OCC.BRepFilletAPI import *
# BRep_Tool забезпечує методи для доступу до геометрії BRep форм
# Наприклад, BRep_Tool_Pnt дозволяє отримати точку з вершини
from OCC.BRep import BRep_Tool_Pnt

# наступний приклад показує як зробити заокруглення ребра, якщо відомі точки його вершин
pt1=gp_Pnt(1,0,0) # перша точка ребра з заокругленням
pt2=gp_Pnt(1,1,0) # друга точка ребра з заокругленням
ex = TopExp_Explorer(solid4, TopAbs_EDGE) # переглядач усіх ребер тіла
mfil=BRepFilletAPI_MakeFillet(solid4) # будівельник заокруглень

while ex.More(): # поки є ще ребро
····e = topods_Edge(ex.Current()) # поточне ребро
····if e.Orientation()==0: # тільки 12 ребер куба будуть переглядатись
```

```
········# ще 12 мають зворотну орієнтацію
········fv=topexp_FirstVertex(e) # перша вершина
ребра
········vp1=BRep_Tool_Pnt(fv) # точка за
вершиною
········lv=topexp_LastVertex(e) # остання
вершина ребра
········vp2=BRep_Tool_Pnt(lv) # точка за
вершиною
········if vp1.IsEqual(pt1,1e-6) and
vp2.IsEqual(pt2,1e-6): # якщо точки рівні
············mfil.Add(0.2, e) # додати
заокруглення до цього ребра
····ex.Next() # перейти до наступного ребра

solid7=mfil.Shape() # тіло з заокругленням
(OCC.TopoDS.TopoDS_Shape)
mch=BRepFilletAPI_MakeChamfer(solid4)
ex = TopExp_Explorer(solid4, TopAbs_FACE) #
переглядач усіх граней тіла
f = topods_Face(ex.Current()) # поточна грань
ex = TopExp_Explorer(solid4, TopAbs_EDGE) #
переглядач усіх ребер грані
e = topods_Edge(ex.Current()) # поточне ребро
mch.Add(0.2,0.3,e,f) # додати фаску
solid8=mch.Shape() # тіло з фаскою

# API для побудови форм шляхом зміщення
from OCC.BRepOffsetAPI import *
edge3=BRepBuilderAPI_MakeEdge(gp_Pnt(),gp_Pnt(0,
1,1)).Edge()
spine=BRepBuilderAPI_MakeWire(edge3).Wire()
edge4=BRepBuilderAPI_MakeEdge(gp_Circ(gp_Ax2(),0
.5)).Edge()
wire2=BRepBuilderAPI_MakeWire(edge4).Wire()
profile=BRepBuilderAPI_MakeFace(wire2).Shape()
```



```
solid9=BRepOffsetAPI_MakePipe(spine,profile).Sha
pe() # витягнути профіль вздовж траєкторії
# Див. також:
# BRepOffsetAPI_MakeThickSolid - створює тіло
шляхом надання товщини поверхням
# BRepOffsetAPI_DraftAngle - уклон плоских,
циліндричних і конічних граней
# BRepOffsetAPI_MakeOffset - зміщення
# BRepOffsetAPI_MakeEvolved - витягування
профілю вздовж траєкторії
# BRepOffsetAPI_ThruSections - створює тіло, яке
проходить через множину січень

# Алгоритми для розрахунку таких глобальних
властивостей як
# довжина, площа, центр мас, об'єм, момент
інерції відносно заданої осі
from OCC.GProp import GProp_GProps
# Функції для розрахунку глобальних властивостей
ліній, поверхонь і тіл
from OCC.BRepGProp import
brepgprop_VolumeProperties
gpro = GProp_GProps()
brepgprop_VolumeProperties(solid9, gpro) #
отримати властивості тіла
com = gpro.CentreOfMass() # центр мас
print com.X(), com.Y(), com.Z()
print gpro.Mass() # об'єм
# закоментуйте/розкоментуйте потрібні команди,
щоб побачити потрібну форму
#display.DisplayShape(solid6)
#display.DisplayShape(solid7)
#display.DisplayShape(solid8)
display.DisplayShape(solid9)
display.FitAll()
start_display()
```

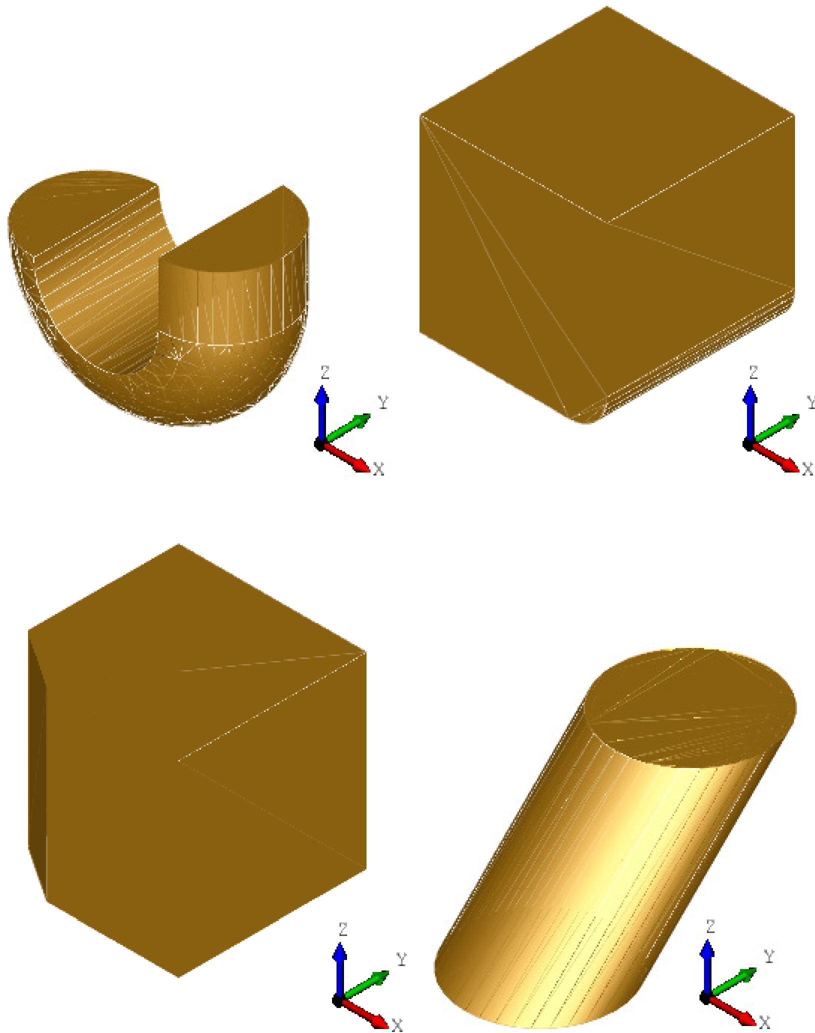


Рисунок 10 - Приклад 12

ПРИКЛАД 13. ВІЗУАЛІЗАЦІЯ ТА GUI

У прикладі test_Display.py продемонстровані засоби для перегляду моделей `Display.OCCViewer` і створення простого інтерфейсу користувача `Display.SimpleGui` (рис. 11).

```
# encoding: utf-8
# Засоби для створення простого GUI
from OCC.Display.SimpleGui import *
# Функції для перегляду моделей
from OCC.Display.OCCViewer import Viewer3d
def HLR(event=None): # відповідає меню HLR
····display.View_Iso() # ізометрія
····display.SetModeHLR() # режим приховування
невидимих ліній
def Shaded(event=None): # відповідає меню Shaded
····s=display.GetSelectedShape() # вибрана мишею
форма
····if s: print s # надрукувати
····display.View_Front() # вид спереду
····display.SetModeShaded() # режим затінення
def WireFrame(event=None): # відповідає меню
WireFrame
····display.DisableAntiAliasing() # відключити
згладжування
····display.SetModeWireFrame() # режим перегляду
каркасу
# об'єкти і функції для роботи з GUI
display, start_display, add_menu,
add_function_to_menu = init_display()
# display - об'єкт класу
OCC.Display.OCCViewer.Viewer3d
display.set_bg_gradient_color(255,255,255,255,25
5,255) # колір фону
#display.SetSelectionModeVertex() # вибір мишею
тільки вершин
```

```
add_menu('View') # створити меню
add_function_to_menu('View',HLR) # створити
підменю
add_function_to_menu('View',Shaded) # створити
підменю
add_function_to_menu('View',WireFrame) #
створити підменю
from OCC.BRepPrimAPI import BRepPrimAPI_MakeBox
s = BRepPrimAPI_MakeBox(10, 20, 30).Shape() #
сторити форму
display.DisplayShape(s,color="black") #
нарисувати форму
from OCC.gp import gp_Pnt
display.DisplayShape(gp_Pnt(),color="black") #
нарисувати точку
# нарисувати текст в заданій позиції
display.DisplayMessage(gp_Pnt(-1,-1,-
1),"0",message_color=(0,0,0))
display.FitAll()
start_display() # розпочати цикл обробки
повідомлень
```

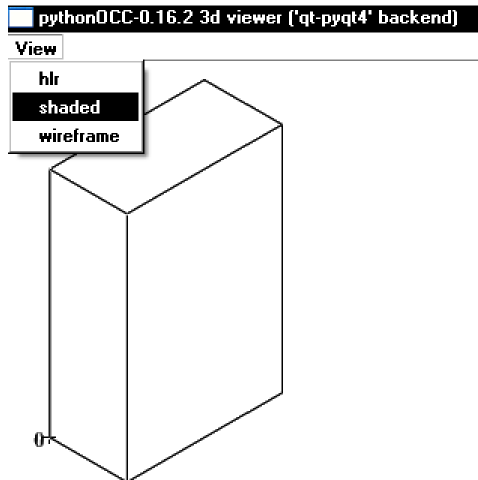


Рисунок 11 - Приклад 13

ПРИКЛАД 14. ІМПОРТ ТА ЕКСПОРТ МОДЕЛЕЙ (STEP ТА STL)

У прикладі test_export.py показані засоби імпорту і експорту моделей для форматів STEP та STL.

```
# encoding: utf-8
from myBaseGeom import *

from OCC.BRepPrimAPI import BRepPrimAPI_MakeBox
solid = BRepPrimAPI_MakeBox(1, 1, 1).Shape() #
призма (TopoDS_Shape)
display.DisplayShape(solid,update=True) #
показати
start_display()

from OCC.STEPControl import STEPControl_Writer,
STEPControl_AsIs, STEPControl_Reader
from OCC.Interface import
Interface_Static_SetCVal
# ініціалізувати експортер STEP
step_writer = STEPControl_Writer()
# AP203: проектування механізмів і загальний 3D
CAD
Interface_Static_SetCVal("write.step.schema",
"AP203")
step_writer.Transfer(solid, STEPControl_AsIs) #
перевести форму у формат STEP
status = step_writer.Write("my.stp") # зберегти
у форматі STEP

step_reader = STEPControl_Reader()
status = step_reader.ReadFile("my.stp") #
прочитати зі STEP
print step_reader.TransferRoot(1)
print step_reader.NbShapes()
```

```
solid = step_reader.Shape(1)

from OCC.StlAPI import StlAPI_Writer,
StlAPI_Reader
StlAPI_Writer().Write(solid, "my.stl") #
зберегти в STL

from OCC.TopoDS import TopoDS_Shape
solid = TopoDS_Shape()
StlAPI_Reader().Read(solid, "my.stl") #
прочитати з STL
```

ПРИКЛАД 15. ІМПОРТ ТА ЕКСПОРТ МОДЕЛЕЙ (BREP)

Приклад occ2freecad.py показує взаємодію FreeCAD і PythonOCC, яка легко реалізується шляхом імпорту і експорту моделей у внутрішньому форматі OCCT BRep.

```
# encoding: utf-8
import sys
# додати шлях до модулів PythonOCC
sys.path.append(r"C:\Python27\Lib\site-
packages")

from OCC.BRepPrimAPI import BRepPrimAPI_MakeBox
box = BRepPrimAPI_MakeBox(10., 20., 30.).Shape()
# створити призму

import Part # модуль Part FreeCAD
# передати форму в FreeCAD
# Part.__fromPythonOCC__(box) # працює не в усіх
версіях PythonOCC (несумісність SWIG)

from OCC.BRepTools import breptools_Write #
функція PythonOCC для запису BRep
```



```
breptools_Write(box,"box.brep") # зберегти в
PythonOCC у форматі BRep
p = Part.read("box.brep") # прочитати в FreeCAD
у форматі BRep
Part.show(p) # показати тіло

p.exportBrep("box.brep") # зберегти в FreeCAD у
форматі BRep

from OCC.TopoDS import TopoDS_Shape
from OCC.BRep import BRep_Builder
from OCC.BRepTools import breptools_Read #
функція PythonOCC для читання BRep
base_shape = TopoDS_Shape()
builder = BRep_Builder()
breptools_Read(base_shape, "box.brep", builder)
# прочитати в PythonOCC у форматі BRep
```

ПРИКЛАД 16. СТВОРЕННЯ ТОПОЛОГІЧНИХ ОБ'ЄКТІВ OCCT ЗА ДОПОМОГОЮ FREECAD-МОДУЛЯ PART

Так само як pythonOCC вільна САПР FreeCAD [19] використовує ядро геометричного моделювання OCCT. Для Python-програмістів FreeCAD може бути використана як набір Python-модулів. На відміну від pythonOCC ці модулі містять тільки основні компоненти OCCT, але їх використання зручніше. Основним модулем для створення і керування BRep-об'єктами є Part. Він містить класи-обгортки над класами OCCT для створення геометричних примітивів (Line, Circle, Arc та ін.) та топологічних форм з базовим класом Part.Shape (Vertex, Edge, Wire, Face, Shell, Solid, CompSolid, Compound).

Для виконання наступного прикладу (freecad_part.py) необхідна установлена FreeCAD 0.16, яка містить свій інтерпретатор Python. Виконайте приклад з консолі так:

```
"c:\Program Files\FreeCAD 0.16\bin\python.exe"
freecad_part.py
```

Для виконання прикладу з довільного інтерпретатора Python 2.7 необхідно присвоїти вірне значення змінній FREECADPATH та ввести в консолі:

```
c:\Python27\python.exe freecad_part.py
```

```
# -*- coding: utf-8 -*-
import sys
FREECADPATH = "c:\\Program Files\\FreeCAD
0.16\\bin"
sys.path.append(FREECADPATH) # шлях до бібліотек
FreeCAD
import math
import FreeCAD # модуль для роботи з програмою
import FreeCADGui # модуль для роботи з GUI
App=FreeCAD
Gui=FreeCADGui
import Part # workbench-модуль для створення і
керування BRep об'єктами

v1=App.Vector(0,0,0) # вектор (або точка)
v2=App.Vector(0,10,0)
v3=App.Vector(5,5,0)
l1=Part.Line(v1,v2) # лінія
a1=Part.Arc(v1,v3,v2) # дуга за трьома точками
e1=l1.toShape() # ребро
# або
#e1=Part.makeLine((0,0,0),(0,10,0)) # ребро
e2=a1.toShape() # ребро
# або
#e2=Part.makeCircle(5,App.Vector(0,5,0),App.Vect
or(0,0,1),-90,90)
bs=Part.BSplineCurve() # B-сплайн
bs.interpolate([(0,0,0),(0,1,1),(0,-1,2)]) #
шляхом інтерполяції
```



```
# або
#bs.approximate([(0,0,0),(0,1,1),(0,-1,2)]) #
шляхом апроксимації
#bs.buildFromPoles([(0,0,0),(0,1,1),(0,-1,2)]) #
за списком полюсів
e3=bs.toShape() # ребро
w1=Part.Wire([e1,e2]) # цикл (сукупність ребер)
f1=Part.Face(w1) # грань
trsf=App.Matrix() # матриця трансформації
trsf.rotateZ(math.pi/4) #повернути навколо осі z
trsf.move(App.Vector(5,0,0)) # перемістити
f2=f1.copy() # копія форми
f2.transformShape(trsf) # виконати трансформацію
# або
# f2.rotate(App.Vector(0,0,0),App.Vector(0,0,1),180.0/4)
# f2.translate(App.Vector(5,0,0))
s1=f2.extrude(App.Vector(0,0,10)) # тіло шляхом
видавлювання
s2=Part.Wire([e3]).makePipe(f1) # тіло шляхом
видавлювання по траєкторії
# або
#s2=Part.Wire([e3]).makePipeShell([w1],True,True
)
s3=f1.revolve(v1,App.Vector(0,1,0),90) # тіло
шляхом обертання
s2=s2.fuse(s3) # об'єднання тіл
# Див. також common, cut, oldFuse
s2=s2.removeSplitter() # видалити непотрібні
ребра (refine shape)
# Див. також makeBox, makeCylinder, makeLoft,
makeThickness, ...
s1=s1.makeFillet(1,[s1.Edges[1]]) #
заокруглення. Див. також makeChamfer

print s1.ShapeType # тип форми
```

```
print s1.Volume # об'єм. Див. також Length,
Area, CenterOfMass
print s1.distToShape(s2) # мінімальна відстань
до іншої форми
print s1.Faces # список граней
print s1.Edges # список ребер
print type(s1.Edges[0].Curve) # тип кривої
першого ребра
print s1.Vertexes[0].Point.x # координата x
точки першої вершини

s1.exportBrep("my.brep") # експорт у форматі
BREP
s1 = Part.Shape()
s1.read("my.brep") # імпорт у форматі BREP
# див. також exportStep, exportIges

# Наступні команди потрібні тільки для
візуалізації створених форм
Gui.showMainWindow() # показати головне вікно
doc=App.newDocument() # створити новий документ
# показати форми
#doc.addObject("Part::Feature","Line").Shape=l1.
toShape()
# або
Part.show(l1.toShape())
Part.show(a1.toShape())
Part.show(w1)
Part.show(f1)
Part.show(f2)
Part.show(s1)
Part.show(bs.toShape())
Part.show(s2)

doc.recompute() # перебудувати
Gui.exec_loop() # головний цикл програми
```


ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Ли К. Основы САПР (CAD/CAM/CAE). - СПб.: Питер, 2004. - 560 с.: ил. ISBN 5-94723-770-9.

2 Голованов Н. Н. Геометрическое моделирование.— М.: Издательство Физико-математической литературы, 2002.—472 с.—ISBN 5-94052-048-0.

3 Max K. Agoston. Computer Graphics and Geometric Modelling: Mathematics. Springer Science & Business Media, 2005. ISBN 978-1-85233-817-6.

4 Max K. Agoston. Computer Graphics and Geometric Modelling: Implementation & Algorithms. Springer Science & Business Media, 2005. ISBN 978-1-84628-108-2.

5 Ian Stroud. Boundary Representation Modelling Techniques. Springer-Verlag London Limited, 2006. - 787 p. - ISBN-13: 978-1-84628-312-3.

6 Erich Hartmann. Geometry and Algorithms for Computer Aided Design. Department of Mathematics Darmstadt University of Technology, October 2003. - 160 p.

7 Duncan Marsh. Applied Geometry for Computer Graphics and CAD.—2nd ed. Springer-Verlag London Berlin Heidelberg, 2005. - 350 p. - ISBN 1852338016.

8 Математика и САПР: В 2-кн. Кн. 1. Пер. с франц./ Шенен П., Коснар М., Гардан И. и др. - М.: Мир, 1988. - 204 с., ил. - ISBN 5-03-000417-3.

9 Норенков И. П., Маничев В. Б. Основы теории и проектирования САПР: Учеб. для втузов по спец. "Вычислительные маш., компл., сист. и сети". - М.: Высш. шк., 1990. - 335 с.: ил. - ISBN 5-06-000730-8.

10 OPEN CASCADE [Electronic resource]. – Mode of access: http://www.opencascade.com

11 Falck, Daniel; Collette, Brad (2012): FreeCAD [How-to]. Solid Modeling with the Power of Python, Packt Publishing, Birmingham, ISBN 978-1-84951-886-4.

12 SALOME - the Open Source Integration Platform for Numerical

Simulation http://www.salome-platform.org.
13 Бизли, Д. Python. Подробный справочник / Дэвид Бизли. - СПб.: Символ-Плюс, 2010. - 864 с.
14 Марк Саммерфилд. Python на практике. / Пер. с англ. Слинкин А.А. - М.: ДМК Пресс, 2014. - 338 с.: ил.
15 PythonOCC: 3D CAD/CAE/PLM development framework for the Python programming language http://www.pythonocc.org.
16 Open CASCADE Technology 6.8.0: Foundation Classes user guides. November 7, 2014.
17 Open CASCADE Technology 6.8.0: Modeling Data user guides. November 7, 2014.
18 Open CASCADE Technology 6.8.0: Modeling Algorithms user guides. November 7, 2014.
19 FreeCAD: An open-source parametric 3D CAD modeler [Electronic resource]. – Mode of access: http://www.freecadweb.org/